Time- and Space-efficient Error Calculation for Multiresolution Direct Volume Rendering

Attila Gyulassy[1], Lars Linsen[1,2], and Bernd Hamann[1]

[1] Institute for Data Analysis and Visualization (IDAV)
University of California, Davis
Davis, CA 95616, U.S.A.*
[2] Department of Mathematics and Computer Science
Ernst-Moritz-Arndt-Universität Greifswald
Greifswald, Germany.**

Summary. Multiresolution data representations are crucial for viewing large volumetric datasets interactively. When data is too large to fit into texture memory, or into main memory, a "cut" must be made through the multiresolution data hierarchy to attain a subset of the data that satisfies the memory requirements. Ideally, a subset is chosen such that the error made when visualizing the subset (compared to a visualization of the full data set) is smaller than that of any other subset of the same size. For real-time applications it is computationally too expensive to calculate the exact error during runtime. Futher, computing error in a pre-processing step is usually not practical due to a large number of possible different configurations each requiring its own error computation. For example, when coupling a multiresolution representation with a direct volume rendering technique, screen-space error depends on the transfer function and viewing direction, making impossible its pre-computation. We present an algorithm that stores an intermediate form of the error, which allows us to approximate screen-space error efficiently. The input for our algorithm is any spatially subdivided multiresolution representation of grid-aligned scalar or multi-variate volume data. We focus on octree- and wavelet-based multiresolution techniques. For each level in the multiresolution hierarchy, the algorithm estimates screen-space error "on the fly," with respect to the current transfer function and viewing direction. The error is approximated by means of a two-dimensional histogram of error pairs. We have extended previous methods by presenting an approach that balances computational and memory costs with approximation quality of the error estimate.

1 Introduction

Visualization of volumetric datasets is a common task used in many fields, including medicine, physics, and other sciences. Complexity is introduced in

*aggyulassy@ucdavis.edu, hamann@cs.ucdavis.edu
**linsen@uni-greifswald.de

this task by the fact that datasets sometimes are too large to fit into texture memory, or main memory. Therefore, data reduction schemes are required to enable interactive exploration and visualization. We refer to the subset that is selected as a "cut" of the data structure. Three steps summarize the main tasks involved in the visualization of large data using multiresolution approximation methods: (i) Compress the original data to a more manageable size; (ii) select the optimal cut such that it minimizes screen-space error while maintaining interactive exploration; and (iii) render the cut in an efficient way. For interactive visualization, frame-rates of at least ten frames per second are desirable. Therefore, the algorithm that selects the cut must be efficient. Also, the algorithm must not have a large memory overhead, since the primary goal is to use available memory to attain as high-quality a representation of the original data as possible. Furthermore, interactive modification of the transfer function is desirable.

Several studies have shown that it is possible to compress large datasets and thereby reduce the I/O and memory footprint. Nguyen and Saupe [8] showed that it is possible to attain quality compression of volumetric datasets using blockwise wavelet representations. Guthe and Straßer [2] demonstrated that using such compression methods and graphics hardware, it is possible to render large datasets at interactive frame rates. Unlike these methods, however, our algorithm can be applied to any data representation, including compressed representations, as long as the representation is hierarchical/nested, i. e., it maintains the property that high-resolution levels are spatially contained in low-resolution levels. Such representations include tree- and wavelet-based structures.

Selection of the cut is important when using multiresolution representations, since different cuts yield different screen-space errors when applying visualization methods. If the error associated with a node in the representation is known, then it is possible to make a decision about the importance of refining the resolution of that node. We present an algorithm that estimates screen-space error efficiently and supports interactive modification of the transfer function. The key observation motivating the algorithm presented in this paper is that there exists an intermediate form of the error that can be exploited to make possible lazy evaluation of the actual screen-space error. The intermediate form of the error can be computed independently of the chosen transfer function and viewing direction. Specifically, instead of storing the actual error (in color space) associated with each node in the data hierarchy, we store a pre-calculated histogram of values and deviations, such that the error can be reconstructed for any transfer function and viewing direction without processing the entire dataset again.

We show that with our algorithm one can attain substantially higher framerates for a guaranteed error bound by minimizing the size of the cut required for that error. Alternately, given a fixed amount of memory, we show that our algorithm selects a near-optimal cut for minimizing screen-space error.

2 Previous Work

The algorithm presented here combines techniques described by Guthe and Straßer [2] and LaMar et al. [6]. LaMar et al. presented an algorithm that makes use of the fact that there are fewer unique error pairs in a large data set than occurrences of such pairs. An error pair (a, b) is a pair of values out of the range of the considered data field. (An error pair (a, b) occurs when value a is used instead of the correct value b). Their algorithm considers byte datasets that have the property that there exist only 256 distinct values and $256^2 = 2^{16}$ possible error pairs, whereas in a data set over a uniform rectilinear grid, e. g., of size 512^3, there are already $512^3 = 2^{27}$ entries of data values. Therefore, storing a 2D table for each non-leaf node in the representation, where the table contains for each error pair (a, b) entries

$$\text{“}Q(a, b) = \text{number of times error pair } (a, b) \text{ occurs,”}$$

makes calculation of the actual error at runtime faster. Several optimizations were introduced to reduce the size of this table, including halving the size of the table by reflecting with respect to the table's diagonal, and run-length encoding. Even though this method provides a fast method for recalculating error, it introduces large storage overhead, as each non-leaf node in the hierarchy has to contain a large data structure representing this table. In addition, there is a high cost of calculating the error at each node, since the entire table must be traversed. Also, this method is restricted to byte datasets. Previous studies [5, 4] developed the error metrics necessary for error calculation.

Guthe and Straßer [2] took a different approach to calculating the error for each node in the hierarchy. In their algorithm, each node in the data hierarchy stores only a small histogram of the maximum deviation for each value in that node. Furthermore, the method bins values into eight groups. Therefore, instead of dealing with a 256^2 table, a single eight-entry array is used. In reconstructing the error, however, all possible combinations of the values must be considered to find a conservative estimate of the error. In practice, the n^2 complexity of this operation can be avoided by storing another small table of the “maximum” and “minimum colors” and opacities for each bin. While this approach is computationally inexpensive and memory-efficient, it creates an overly conservative estimation of error. Indeed, the maximum color is compared with the minimum color of every other element, and then scaled by the number of items binned, which greatly overshoots the actual error. Therefore, this method overestimates the error associated with a node in such a way that it could produce an inferior cut of the multiresolution representation.

We present a new, hybrid approach, which uses a histogram similar to the one used by LaMar et al. [6], but also applies a binning procedure, such as described by Guthe and Straßer [2], see Figure 1. In this way, we obtain a closer approximation of the actual error, while keeping the data structure and

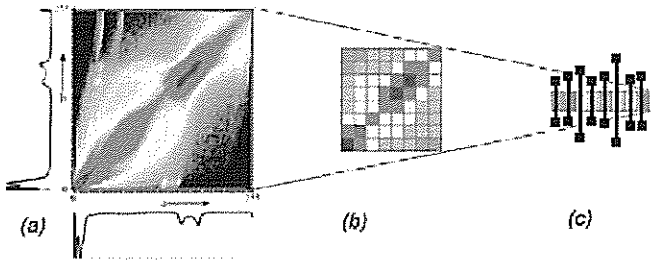

Fig. 1. For each node in the data hierarchy, (a) shows the table generated by LaMar et al. [6], (c) shows the binning method of Guthe and Straßer [2], and (b) shows how our algorithm combines the two methods.

computation overheads small. Furthermore, our binning approach enables us to work on any multi-variate dataset, not only on byte data.

Other methods [1, 3, 11] can be used to calculate the error associated with levels in a hierarchy. However, these methods rely on a fixed transfer function. Therefore, whenever the transfer function is modified, the entire dataset must be traversed. This characteristic prohibits interactive modification of the transfer function.

3 Error Estimation

Our algorithm presented here utilizes a conservative error estimation to select the cut through the multiresolution representation of a dataset. The cut consists of the nodes that are kept in memory for the purpose of rendering. The error can be measured by the root-mean-square (RMS) difference between the images generated by rendering the actual and approximating data, called the screen-space error. The error in a single pixel is defined as

$$error_{color}(x) = \left| \int_0^d Opacity_h(x) * Color(Value_h(x))\, dx - \int_0^d Opacity_l(x) * Color(Value_l(x))\, dx \right| .$$

The function $Opacity_h$ refers to the function that defines the progressive visibility along the ray, parameterized by x for the high-resolution data representation; $Value_h$ refers to the value returned by interpolation of the high-resolution data. Similarly, $Opacity_l$ and $Value_l$ refer to the corresponding functions for the low-resolution data. The integral is evaluated over the interval $(0, d)$, where d is the far cut plane.

In our conservative estimate, we approximate this value by the integral of the errors along the ray projected from that pixel convolved with the progressive opacity function, i.e.,

$$error_{approx}(x) = \int_0^d |Opacity_h(x) * Color(Value_h(x)) - Opacity_l(x) * Color(Value_l(x))| \; dx \; .$$

Considering the triangle inequality, $error_{color}(x) \leq error_{approx}(x)$.

We further simplify this equation by using the opacity in each node instead of the progressive opacity, so that the error contributed by each node is view-independent. Therefore, a conservative estimate of the error contributed by each node is sufficient to compute a conservative estimate of the final screen-space error of the image. The error of a particular node in the cut is also useful for determining whether or not to refine the cut at that node.

3.1 View-independent Error at a Node

For simplicity, we first only consider a piecewise-constant interpolation of the data. Also, we consider only one color channel at a time so that the transfer function is scalar-valued. We combine the error contributions of each channel after they have been computed independently. As is done by LaMar et al. [6], we can use two different error norms for calculating the absolute error at a node in the data representation. The L_∞-error defines the error as the maximum of the errors under that node, i.e.,

$$error_{L_\infty} = \max_{p \in B} \{|Color(Value_h(p)) - Color(Value_l(p))|\} \; , \qquad (1)$$

where p refers to points of the original data that reside inside node B of the hierarchy. This error calculates the largest deviation in color that can occur inside a node. The RMS error averages the errors under that node, i.e.,

$$error_{RMS} = \sqrt{\frac{1}{n} \sum_{p \in B} (Color(Value_h(p)) - Color(Value_l(p)))^2} \; , \qquad (2)$$

where n is the total number of points $p \in B$. For simplicity, we limit our discussion to $error_{RMS}$. To calculate this error, however, we need to use the entire data structure. We can rewrite this error as

$$error_{RMS} = \sqrt{\frac{1}{n} \sum_{i} (Color(a_i) - Color(b_i))^2 * Q(a_i, b_i)} \; , \qquad (3)$$

where $\{(a_i, b_i)\} = \{(Value_h(p_i), Value_l(p_i)) | p_i \in B\}$ and $Q(a_i, b_i)$ is the number of times that the error pair (a_i, b_i) appears inside node B. In the case of byte data, it is possible to represent Q explicitly as a table that is significantly smaller than n for blocked data, and we can efficiently compute the error at a

node using this histogram. However, for real-valued data, no guarantees can be made concerning the number of unique error pairs (a_i, b_j). Therefore, the size of Q is only bounded by n. Straightforward storage of the table Q would be inefficient. Instead, we fix a histogram size for Q, binning error pairs. Q represents a table of bins, each bin (a_i, b_j) counting the number of occurrences of error pairs in its range. We reconstruct a conservative error efficiently from this histogram.

At each node in the hierarchy, we store such a table Q of error pairs. We define the table Q at node B as $Q(a_i, b_j) = m$, where m is the number of points $p \in B$ such that $Value_l(p) \in [a_i, a_{i+1})$ and $Value_h(p) \in [b_j, b_{j+1})$. Each bin in the histogram stores the number of occurrences of error pairs whose values fall within a range of values. Figure 2 shows how this table is created for the lowest non-leaf node in the multiresolution hierarchy.

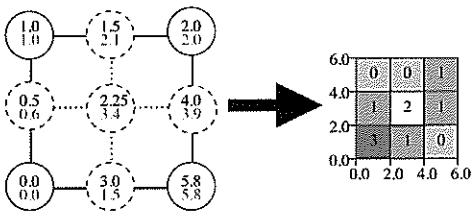

Fig. 2. Two levels in a quad-tree hierarchy. Data points (left) display low-resolution $Value_l(p)$ on top, and high-resolution $Value_h(p)$ on the bottom, and the two values together form an error pair. A 3×3 histogram (right) associated with the low-resolution node stores the number of ocurrences of error pairs. We assume piecewise-constant interpolation, with the data ranging in value in the interval [0.0, 6.0].

To compute the error at a given node B, we reference the table Q of error pairs associated with that node. We use the form of the error given by Equation 3 to generate our error formula as

$$error_{RMS} = \sqrt{\frac{1}{n} \sum_{i,j} (MaxError([a_i, a_{i+1}), [b_j, b_{j+1})))^2 * Q(a_i, b_j)} \ , \quad (4)$$

where $MaxError([a_i, a_{i+1}), [b_j, b_{j+1}))$ is the largest possible error in color that can occur in the intervals $[a_i, a_{i+1})$ and $[b_j, b_{j+1})$. We apply the transfer function to compute the maximum and minimum colors for each interval. We define $MaxError$ as

$$\begin{aligned} MaxError([a,b),[c,d)) = &\max(Color([a,b)) \cup Color([c,d))) \\ &- \min(Color([a,b)) \cup Color([c,d))) \ . \end{aligned} \quad (5)$$

The variables a, b, c, and d represent values in the domain of the transfer function. Here, $Color([a,b))$ returns a range of color values in the interval

$[a, b)$. Figure 3 illustrates $MaxError$. The transfer function is piecewise-linear. Therefore, all extrema occur either at the endpoints of the intervals, or at data points of the transfer function. We extract extreme values by sweeping through the transfer function inside the intervals $[a_i, a_{i+1})$ and $[b_j, b_{j+1})$. The maximal error is computed as the difference between the maximum color value and the minimum color value. We scale the maximum error in a node by the maximum distance inside the node to obtain a conservative estimate of the error. The maximum distance in a node in a standard rectilinear grid is the diagonal length $\sqrt{3}$.

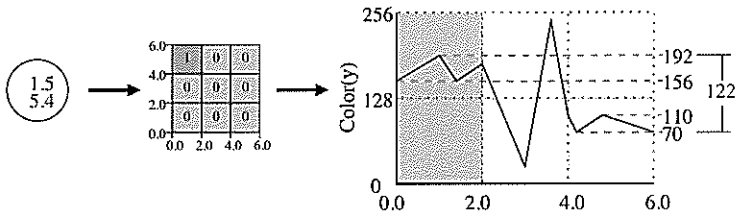

Fig. 3. Error pairs (left) are used to populate the histogram (middle). To reconstruct the error associated with bin $[0.0, 2.0) \times [4.0, 6.0)$ we determine the maximally possible difference in those ranges in the transfer function (right), $MaxError() = 122$. We assume piecewise-constant interpolation of the data.

The error formulation in Equation 6 is defined for piecewise-constant interpolation applied to a dataset. To maintain a conservative error estimate when using trilinear interpolation, the histogram Q of a node B must be modified. $Q(a_i, b_j)$ represents the number of nodes B_i contained in B, where the minimum function value is in the range a_i, and the maximum function value is in the range b_j. To reconstruct the error, $MaxError$ now returns the maximum color difference in the range $[min(a_i, b_j), max(a_i, b_j)]$. When the dimension of Q is $N \times 1$, this method reduces to the one presented in [2]. For our results, we have used this formulation of error.

The calculation of the maximum error of a bin is an expensive operation. However, a major improvement can be made when each table Q has the same range and size at every node. In this case, a single 2D table can be calculated representing the maximum difference between colors for each bin in the histogram Q. Therefore, the (i, j)-th element holds the maximum color difference for a bin (a_i, b_j) in table Q. This drastically reduces the amount of calculation necessary for each node, since this only needs to be calculated once whenever the transfer function is modified. The result is stored in a table for subsequent look-ups.

Increasing histogram size improves the accuracy of the error estimation, as it reduces the range of each bin, and therefore the maximum error associated with that bin. Table 1 shows the space overhead associated with different his-

Histogram size	2^2	8^2	64^2	256^2
32^3	0.012%	0.195%	12.500%	200.000%
64^3	0.002%	0.024%	1.560%	25.000%
128^3	<0.001%	0.003%	0.195%	3.120%
256^3	<0.001%	<0.001%	0.024%	0.391%

Table 1. Increasing histogram size leads to larger memory overhead associated with multiple block sizes. The numbers are the percentages of the total memory used to store histograms.

togram sizes. The maximum acceptable histogram size at each node depends on the size of the blocks, and also on data size. A large histogram can lead to severe storage overhead. Indeed, for efficient estimation of the error, it is important to keep the error estimation structure in memory. Therefore, the size of the histogram must be balanced with performance considerations.

3.2 View-dependent Error

Another consideration in calculating error is visibility of a node. Two factors contribute to visibility: projected solid angle and opacity.

Projected solid angle refers to the amount of screen space that a node and its sub-hierarchy occupy when projected. When a node occupies less than a pixel of screen space, its screen-space error is very small. Conversely, when the screen space of a node is large, even smaller errors are noticed. Therefore, a new view-dependent error function can be defined as

$$error_{node} = error_{RMS} \cdot \phi \ ,$$

where ϕ is the projected solid angle of the node. Since the actual value of the projected solid angle is expensive to calculate, we approximate it by using the distance d from the camera to the node, i.e.,

$$error_{node} = error_{RMS} \cdot \alpha \cdot \frac{r^2}{d^2} \ ,$$

where r is the maximum radius of the node and α is a constant.

Opacity is difficult to calculate efficiently, especially since it relies on the transfer function and on the viewing direction. The error methods discussed so far are used to select the nodes that in the working set, needed for rendering. To calculate opacity, a front-to-back calculation must take place to eliminate nodes that are occluded. One possible way to perform this task, without calculating the entire working set, is to process nodes front-to-back and subdivide them in that order. Therefore, a node is only considered for subdivision according to the previously defined error metric once all nodes in front of it satisfy a particular error condition.

Following this approach, only the nodes in the final working set are considered. Unfortunately, the size of this set is no longer bounded, as there is no limit on the number of subdivision levels of nodes in the front. As a result, the memory bound for the working set may be exceeded with a sub-optimal selection of nodes. Due to the complexity involved in calculating opacity while selecting a cut of the multiresolution representation, occlusion culling is usually employed only during rendering, once the cut has been selected. Guthe and Straßer [2] used such a method to avoid rendering occluded blocks.

3.3 Cut Selection

Assuming that the error for each node is known, the overall procedure for selecting nodes and rendering performs these steps:

Pre-processing:

1. Initialize multiresolution data structure.
2. Calculate histograms in bottom-up manner.

Runtime:

1. Initialize the cut with the top level node.
2. While space left in memory:
   - Find node with largest error.
   - Subdivide this node and add children to the cut.
3. Render the cut.

The greedy algorithm selects the node with the largest error, and subdivides it. As a termination condition we either use the amount of free memory left or a particular error threshold not to be exceeded. In either case, the algorithm terminates, since space consumption increases and error decreases as we refine the cut.

4 Multiresolution Representation

Our algorithm can be applied to any nested multiresolution representation, i. e., a representation that satisfies the multiresolution analysis criteria presented by Rodler [10]. In particular, the multiresolution representation must subdivide the entire data space, with nodes at each higher level in the representation spatially bounding their child nodes. Furthermore, the accuracy of the representation must not decrease as the cut is refined. Therefore, the error of each child of a node must be smaller than or equal to the error at that node.

There are many representations that satisfy these requirements. Some commonly used ones are octrees and wavelet-based representations. For simplicity, our algorithm was implemented using an octree representation.While octrees

are attractive due to their simplicity, several studies have shown that wavelet-based representations are efficient as well. In particular, when using wavelets, it is possible to compress the original data, to alleviate some of the difficulties in large dataset rendering. Haar wavelets are the most commonly used wavelets. Park and Ihm [9] attained both compression and increased performance by using that representation. Indeed, Rodler [10] presented several more complicated wavelet transforms. However, the Haar wavelet transform is the most appropriate method for most multiresolution techniques, due to its simplicity and efficiency.

5 Rendering

Rendering of volumetric data is a well studied topic. Very large datasets, however, pose additional challenges. Levoy [7] implemented a scheme for interactive raycasting. However, his method involves massively parallel rendering. Still, the optimal performance for raycasting a 512^3 dataset with 96 processors was less than two frames per second. Although CPU performance has increased dramatically over recent years, straightforward raycasting of large datasets is not practical for interactive visualization on PCs.

Some of the fastest techniques for rendering volumetric data on a standard PC utilize 3D texture hardware. LaMar et al. [5] showed how to improve image quality by using object-aligned slices. Westermann [11] showed that it is possible to use texture hardware for multiresolution representations. Guthe and Straßer [2] attained about ten-frames-per-second performance for a 40GB dataset, using their error estimation technique combined with texture hardware. Therefore, once errors are calculated and a cut is determined, any of the previous techniques could be used to render the data.

6 Results and Discussion

Our algorithm provides an improved method for estimating screen-space error associated with levels in a multiresolution data representation. Our method balances storage overhead, processing time, and quality of estimation. Using this improved selection strategy, we pick a better cut through the multiresolution representation, meaning that the same-sized cut yields lower error.

In Section 3, we showed that the storage overhead for this method is dependent on the size of the histograms used and the size of the dataset. Therefore, we can balance speed of the algorithm with the memory footprint.

Figure 4 shows that, given an error threshold, increasing the histogram size estimates the error of a cut with higher accuracy. In this case, we wish to find the smallest cut such that the error threshold is satisfied. For a "nice" dataset, i.e., a data set with smoothly changing function values, the error approximation improves with size of the histogram in a reasonable manner. However,

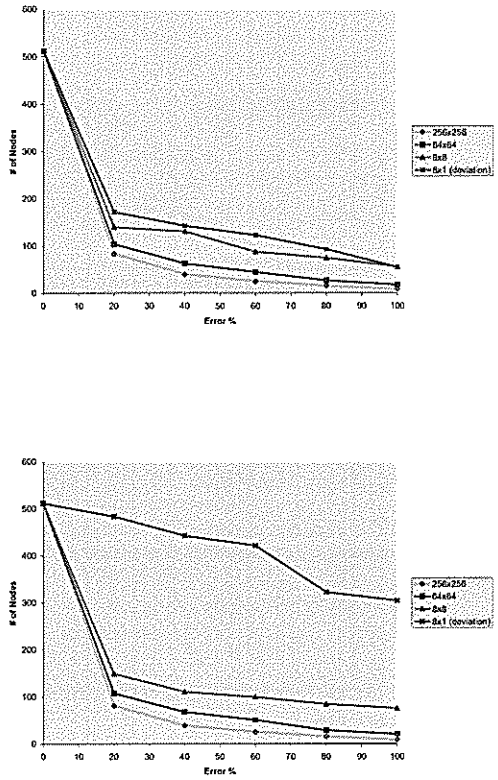

Fig. 4. Increasing error tolerance decreases the size of the cut needed to satisfy the error condition. Error is measured as percentage of the maximum error, which is the error associated with the coarsest resolution. "8x1(deviation)" refers to the algorithm presented by Guthe and Straßer [2]. Left: Application to "nice" artificial dataset consisting of distances to points distributed in the domain. Right: Application to same dataset with some sparse noise inserted.

with sparse noise inserted into the dataset, we see a substantial improvement with increased histogram size. One advantage of our method is that small perturbations in the dataset do not significantly increase the estimated error, in contrast to the algorithm presented in Guthe and Straßer [2].

We performed our analysis for the human skull dataset, which has size 256^3 and integer data values in the range $[0, 255]$. This dataset has both high-frequency and low-frequency regions, and therefore is suitable for analysis purposes. Results were generated with a 2GHz Pentium 4 processor with 512Mb of main memory. The dataset was divided into 32^3 blocks and rendered using a straightforward raycasting method. Recalculation of the error when changing the transfer function required less than one milli-second. The recalculation of the error does not scale with dataset size, it scales only only with the size of the blocks and the size of the cut. Therefore, interactive modification of the transfer function is possible.

As expected, increasing the size of the cut improves the accuracy of the final image. As more data is used, the error decreases. However, the real benefit of this selection strategy is this: When the cut size is held constant, using a larger histogram reduces the final error by selecting a better cut. Figure 5 shows the inverse relationship between histogram size and screen-space error.

7 Conclusions and Future Work

We have presented a cut selection strategy for multiresolution direct volume rendering of large data that supports interactive modification of a transfer function. We have improved previous methods [2, 6], since our method can deal with any scalar-valued dataset, using an improved error estimation scheme. Our histogram approach is memory-efficient and can be used in any multiresolution direct volume rendering method. An important property of our approach is that the size of a histogram determines accuracy of rendering results. Therefore, we can balance computational and memory costs with quality. Although the histograms we showed in the previous section are square, it is possible to attain good results with non-square histograms. Our results suggest that it is possible to tune the histogram size automatically for a particular architecture and dataset size.

We plan to extend this algorithm to calculate errors for time-varying volumetric data. Another possible future research area is error calculation for vector fields.

Acknowledgments

This work was supported by the National Science Foundation under contract ACI 9624034 (CAREER Award), through the Large Scientific and Software Data Set Visualization (LSSDSV) program under contract ACI 9982251,

through the National Partnership for Advanced Computational Infrastructure (NPACI) and a large Information Technology Research (ITR) grant; and the National Institutes of Health under contract P20 MH60975-06A2, funded by the National Institute of Mental Health and the National Science Foundation. We thank the members of the Visualization and Computer Graphics Research Group at the Institute for Data Analysis and Visualization (IDAV) at the University of California, Davis.

References

1. Imma Boada, Isabel Navazo, and Roberto Scopigno. Multiresolution volume visualization with a texture-based octree. *The Visual Computer*, 17(3):185–197, 2001.
2. Stefan Guthe and Wolfgang Straßer. Advanced Techniques for High-Quality Multi-Resolution Volume Rendering. *Computers & Graphics*, 28(1):51–58, February 2004.
3. Stefan Guthe, Michael Wand, Julius Gonser, and Wolfgang Straßer. Interactive rendering of large volume data sets. In *Proceedings of the conference on Visualization '02*, pages 53–60. IEEE Computer Society, 2002.
4. Eric LaMar, Bernd Hamann, and Kenneth I. Joy. Multiresolution techniques for interactive texture-based volume visualization. In *VIS '99: Proceedings of the conference on Visualization '99*, pages 355–361, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.
5. Eric C. LaMar, Mark A. Duchaineau, Bernd Hamann, and Kenneth I. Joy. Multiresolution techniques for interactive texturing-based rendering of arbitrarily oriented cutting-planes. In W.C. de Leeuw and R. Van Liere, editors, *Proceedings of VisSym 00 The Joint Eurographics and IEEE TCVG Conference on Visualization*, pages 105–114. Springer-Verlag, 2000.
6. Eric C. LaMar, Bernd Hamann, and Kenneth I. Joy. Efficient error calculation for multiresolution texture-based volume visualization. In Gerald Farin, Bernd Hamann, and Hans Hagen, editors, *Hierachical and Geometrical Methods in Scientific Visualization*, pages 51–62, Heidelberg, Germany, 2003. Springer-Verlag.
7. Marc Levoy. Display of surfaces from volume data. *IEEE Comput. Graph. Appl.*, 8(3):29–37, 1988.
8. Ky Giang Nguyen and Dietmar Saupe. Rapid high quality compression of volume data for visualization. *Computer Graphics Forum*, 20(3):C49–C56, Sept 2001.
9. Sanghun Park and Insung Ihm. Wavelet-based 3d compression scheme for interactive visualization of very large volume data. *Computer Graphics Forum*, 18(1), March 1999.
10. Flemming Friche Rodler. Wavelet based 3d compression with fast random access for very large volume data. In *Proceedings of the 7th Pacific Conference on Computer Graphics and Applications*, page 108. IEEE Computer Society, 1999.
11. Rüdiger Westermann. A multiresolution framework for volume rendering. In Arie Kaufman and Wolfgang Krüger, editors, *1994 Symposium on Volume Visualization*, pages 51–58, 1994.

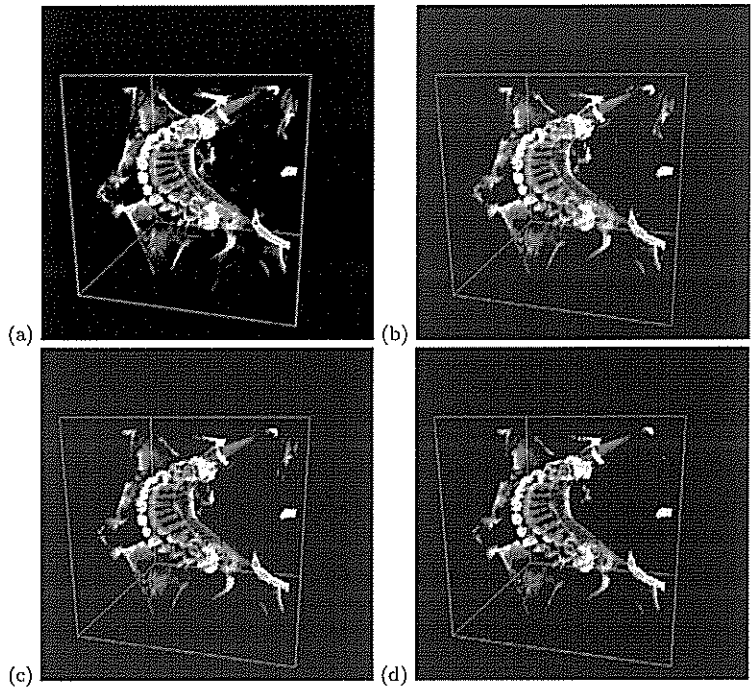

Fig. 5. (a) Dataset at full resolution. Images (b), (c) and (d) show a cut of 63 nodes selected by our strategy using histogram sizes of 64^2, 16^2, and 4^2, respectively.

Histogram size	4^2	16^2	64^2	Guthe and Straßer [2]
RMS error	6.20	5.71	4.68	6.29

Table 2. RMS errors associated with image generated by our algorithm for each histogram size. The last column is the RMS error of the image generated by Guthe and Straßer [2] using eight bins.